Homework 2 Programming Problem 7 (30 points)

Problem Description

In this problem you will implement polynomial linear least squares regression on two datasets, with and without regularization. Additionally, you will use gradient descent to optimize for the model parameters.

Fill out the notebook as instructed, making the requested plots and printing necessary values.

You are welcome to use any of the code provided in the lecture activities.

Summary of deliverables:

Results:

- Print fitted model parameters `w` for the 4 models requested without regularization
- Print fitted model parameters `w` for the 2 models requested *with* L_2 regularization
- Print fitted model parameters `w` for the one model solved via gradient descent

Plots:

- 2 plots of each dataset along with the ground truth function
- 4 plots of the fitted function along with the respective data and ground truth function for LLS without regularization
- 2 plots of the fitted function along with the respective data and ground truth function for LLS with L_2 regularization
- 1 plot of the fitted function along with the respective data and ground truth function for LLS with L_2 regularization solved via gradient descent

Discussion:

- Discussion of challenges fitting complex models to small datasets
- Discussion of difference between the L_2 regularized model versus the standard model
- Dicussion of whether gradient descent could get stuck in a local minimum
- Discussion of gradient descent results versus closed form results

Imports and Utility Functions:

In [75]:
```
import numpy as np
import matplotlib.pyplot as plt

def gt_function():
    xt = np.linspace(0,1,101)
    yt = np.sin(2 *np.pi*xt)
```

```
    return xt, yt

def plot_data(x,y,xt,yt,title = None):
    # Provide title as a string e.g. 'string'
    plt.plot(x,y,'bo',label = 'Data')
    plt.plot(xt,yt,'g-', label = 'Ground Truth')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid()
    plt.legend()
    if title:
        plt.title(title)
    plt.show()

def plot_model(x,y,xt,yt,xr,yr,title = None):
    # Provide title as a string e.g. 'string'
    plt.plot(x,y,'bo',label = 'Data')
    plt.plot(xt,yt,'g-', label = 'Ground Truth')
    plt.plot(xr,yr,'r-', label = 'Fitted Function')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid()
    plt.legend()
    if title:
        plt.title(title)
    plt.show()
```

Load and visualize the data

The data is contained in `d10.npy` and `d100.npy` and can be loaded with `np.load()`.

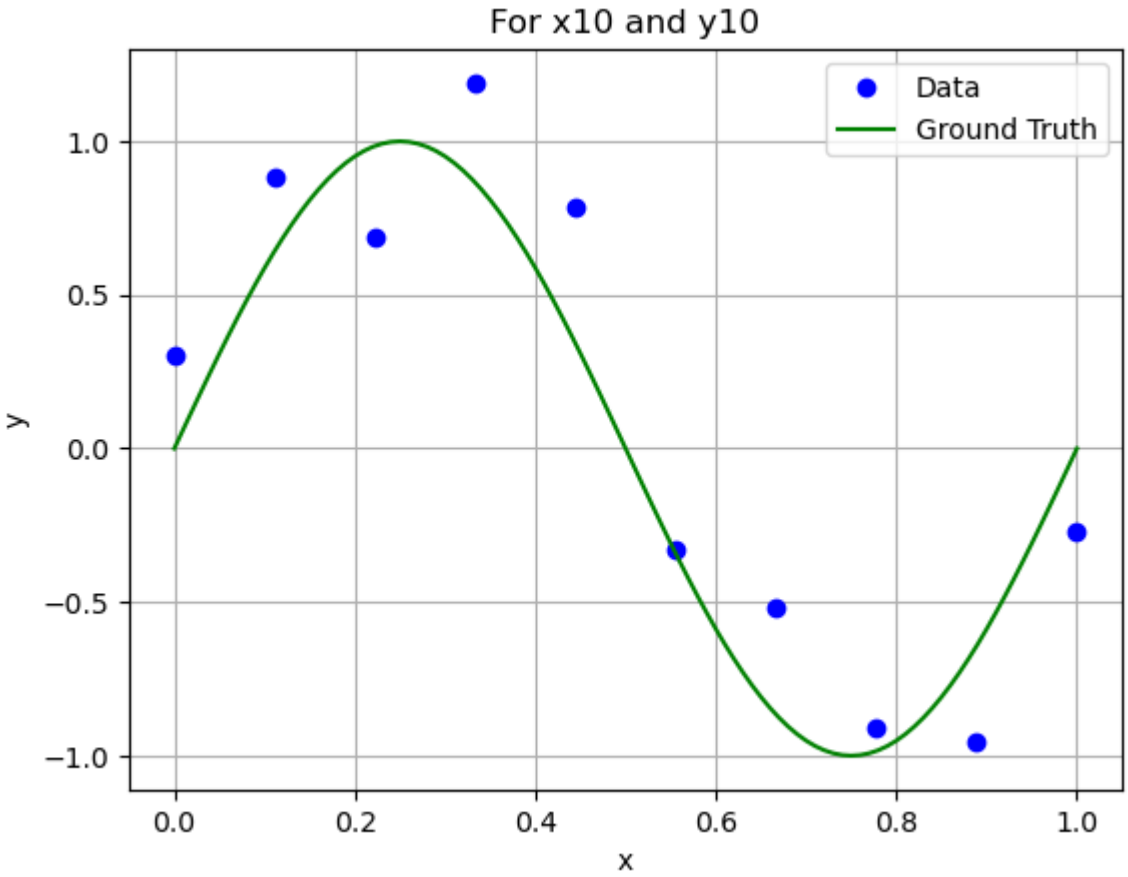
Store the data as:

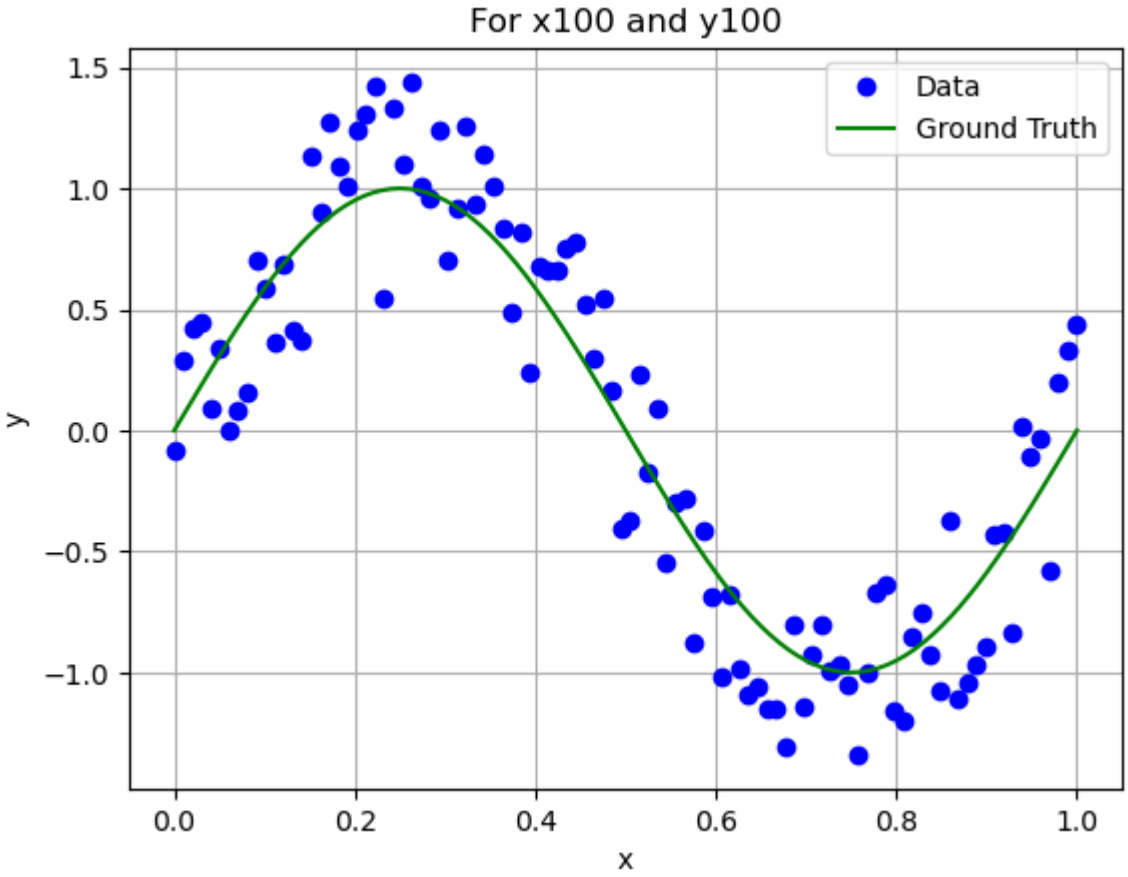
- `x10` and `x100` (the first column of the data)
- `y10` and `y100` (the second column of the data)

Generate the ground truth function $f(x)=\textrm{sin}(2\pi x)$ using `xt, yt = gt_function()`.

Then visualize the each dataset with `plotxy(x,y,xt,yt,title)` with an appropriate title. You should generate two plots.

In [76]:
```
# YOUR CODE GOES HERE
# ground truth for function for d10 and d100
d10 = np.load('d10.npy')
d100 = np.load('d100.npy')
x10 = d10[:,0]
x100 = d100[:,0]
y10 = d10[:,1]
y100 = d100[:,1]
xt,yt = gt_function()
plot_data(x10,y10,xt,yt,"For x10 and y10")
plot_data(x100,y100,xt,yt,"For x100 and y100")
```





Implement polynomial linear regression

Now you will implement polynomial linear least squares regression without regularization using the closed form solution from lecture to compute the model parameters. You will consider the following 4 cases:

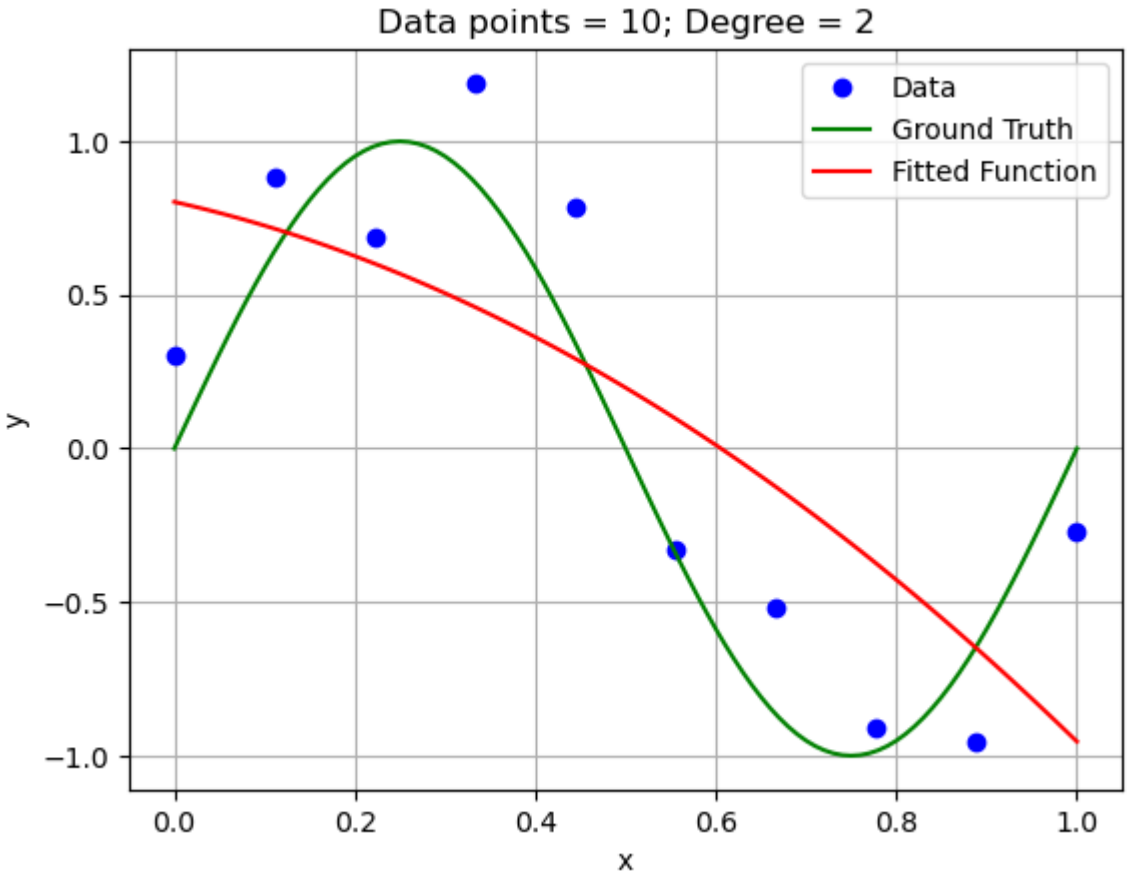
1. Data: data10.txt, Model: 2nd order polynomial (highest power of x in your regression model = 2)
2. Data: data100.txt, Model: 2nd order polynomial (highest power of x in your regression model = 2)
3. Data: data10.txt, Model: 9th order polynomial (highest power of x in your regression model = 9)
4. Data: data100.txt, Model: 9th order polynomial (highest power of x in your regression model = 9)

For each model:

- Print the learned model parameters `w`
- Use the model parameters `w` to predict `yr` values over a range of x values given by `xr = np.linspace(0,1,101)`
- Plot the data, ground truth function, and regressed model using `plot_model(x,y,xt,yt,xr,yr,title)` with an appropriate title.

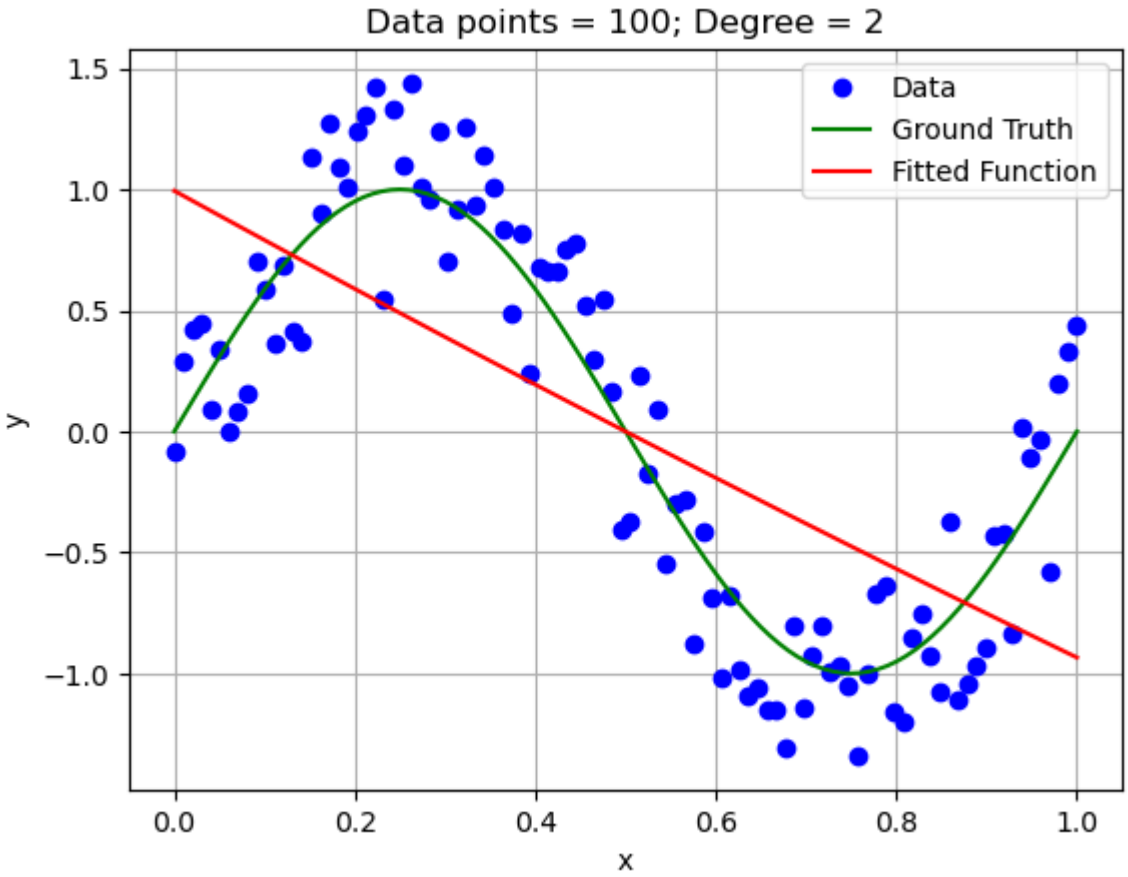
In [77]:
```
# YOUR CODE GOES HERE
x10_sq = x10*x10
X10 = np.array([x10_sq, x10, np.ones_like(x10)]).T

xr_10 = np.linspace(0,1,101)
xr_10_sq = xr_10**2
xr = np.array([xr_10_sq, xr_10, np.ones_like(xr_10)]).T

w = np.linalg.inv(X10.T @ X10) @ X10.T @ y10.reshape(-1,1)
print("Quadratic Coefficients for degree 2 and 10 data points:", w.flatten(),"\n")
yr = xr @ w

plot_model(x10, y10, xt, yt, xr_10, yr, "Data points = 10; Degree = 2")
```

```
Quadratic Coefficients for degree 2 and 10 data points: [-1.09384447 -0.66283292  0.8
0276877]
```



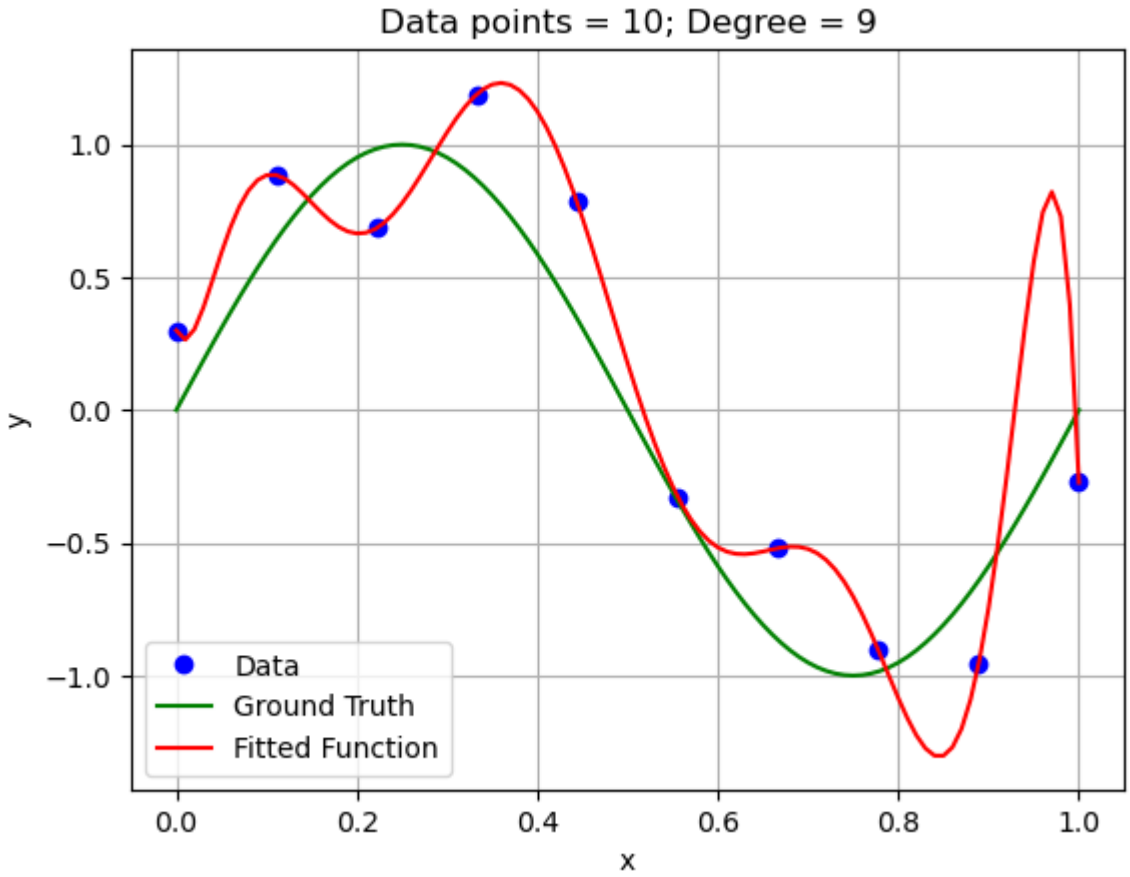
In [78]:
```
x100_sq = x100*x100
X100 = np.array([x100_sq, x100, np.ones_like(x100)]).T

xr_100 = np.linspace(0,1,101)
xr_100_sq = xr_100**2
xr = np.array([xr_100_sq, xr_100, np.ones_like(xr_100)]).T

w = np.linalg.inv(X100.T @ X100) @ X100.T @ y100.reshape(-1,1)
print("Quadratic Coefficients for degree 2 and 100 data points:", w.flatten(),"\n")
yr = xr @ w

plot_model(x100, y100, xt, yt, xr_100, yr, "Data points = 100; Degree = 2")
```

```
Quadratic Coefficients for degree 2 and 100 data points: [ 0.12128272 -2.04993418  0.
99435046]
```



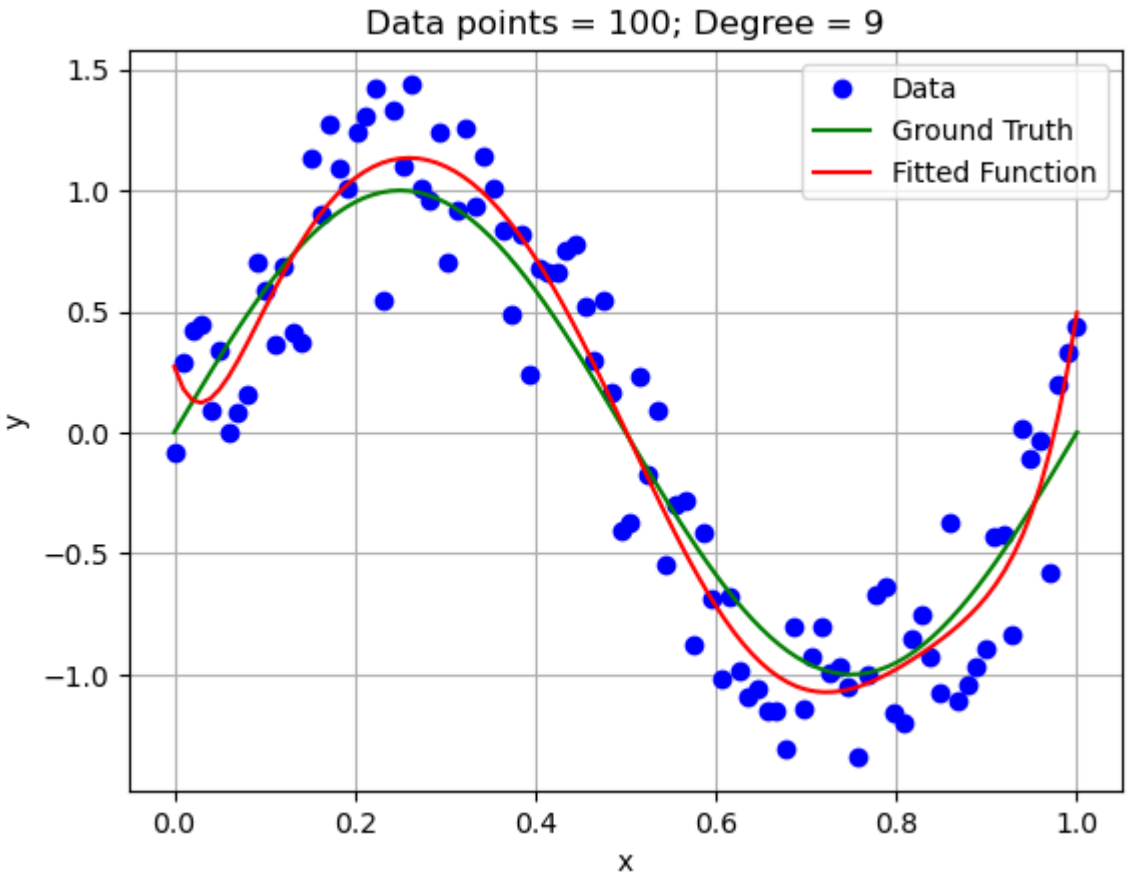
In [79]:
```
x10_2 = x10**2
x10_3 = x10**3
x10_4 = x10**4
x10_5 = x10**5
x10_6 = x10**6
x10_7 = x10**7
x10_8 = x10**8
x10_9 = x10**9
X10 = np.array([x10_9, x10_8, x10_7, x10_6, x10_5, x10_4, x10_3, x10_2, x10, np.ones_]

xr10 = np.linspace(0,1,101)
xr10_2 = xr10**2
xr10_3 = xr10**3
xr10_4 = xr10**4
xr10_5 = xr10**5
xr10_6 = xr10**6
xr10_7 = xr10**7
xr10_8 = xr10**8
xr10_9 = xr10**9
xr = np.array([xr10_9, xr10_8, xr10_7, xr10_6, xr10_5, xr10_4, xr10_3, xr10_2, xr10, r

w = np.linalg.inv(X10.T @ X10) @ X10.T @ y10.reshape(-1,1)
print("Quadratic Coefficients for degree 9 and 10 data points:", w.flatten(),"\n")
yr = xr @ w
plot_model(x10, y10, xt, yt, xr10, yr, "Data points = 10; Degree = 9")
```

```
Quadratic Coefficients for degree 9 and 10 data points: [-3.91859889e+04  1.67639024e
+05 -2.96314110e+05  2.79104415e+05
 -1.50525733e+05  4.63850649e+04 -7.66657156e+03  5.71841525e+02
 -8.51390078e+00  3.00000725e-01]
```



In [80]:
```
x100_2 = x100**2
x100_3 = x100**3
x100_4 = x100**4
x100_5 = x100**5
x100_6 = x100**6
x100_7 = x100**7
x100_8 = x100**8
x100_9 = x100**9
X100 = np.array([x100_9, x100_8, x100_7, x100_6, x100_5, x100_4, x100_3, x100_2, x100,

xr100 = np.linspace(0,1,101)
xr100_2 = xr100**2
xr100_3 = xr100**3
xr100_4 = xr100**4
xr100_5 = xr100**5
xr100_6 = xr100**6
xr100_7 = xr100**7
xr100_8 = xr100**8
xr100_9 = xr100**9
xr = np.array([xr100_9, xr100_8, xr100_7, xr100_6, xr100_5, xr100_4, xr100_3, xr100_2,

w = np.linalg.inv(X100.T @ X100) @ X100.T @ y100.reshape(-1,1)
print("Quadratic Coefficients for degree 9 and 100 data points:", w.flatten(),"\n")
yr = xr @ w

plot_model(x100, y100, xt, yt, xr100, yr, "Data points = 100; Degree = 9")
```

```
Quadratic Coefficients for degree 9 and 100 data points: [-2.21780596e+03  1.15803259
e+04 -2.48707277e+04  2.86950313e+04
 -1.95227568e+04  8.13334405e+03 -2.07030008e+03  2.84964284e+02
 -1.18524605e+01  2.71586959e-01]
```



Discussion:

When the sample size (number of data points) is small, what issues or tendencies do you see with complex models?

When the sample size is small the model is overfitting. The model has issues in making the best fit line as the variance becomes high and it leads to an issue of generalization

Implement polynomial linear regression with L_2 regularization

You will repeat the previous section, but this time using L_2 regularization. Your regularization term should be $\lambda w' \mathbb{I}_m w$, where $\lambda = e^{-10}$, and $\mathbb{I}_m$ is the modified identity matrix that masks out the bias term from regularization.

You will consider only two cases:

1. Data: data10.txt, Model: 9th order polynomial (highest power of x in your regression model = 9)
2. Data: data100.txt, Model: 9th order polynomial (highest power of x in your regression model = 9)

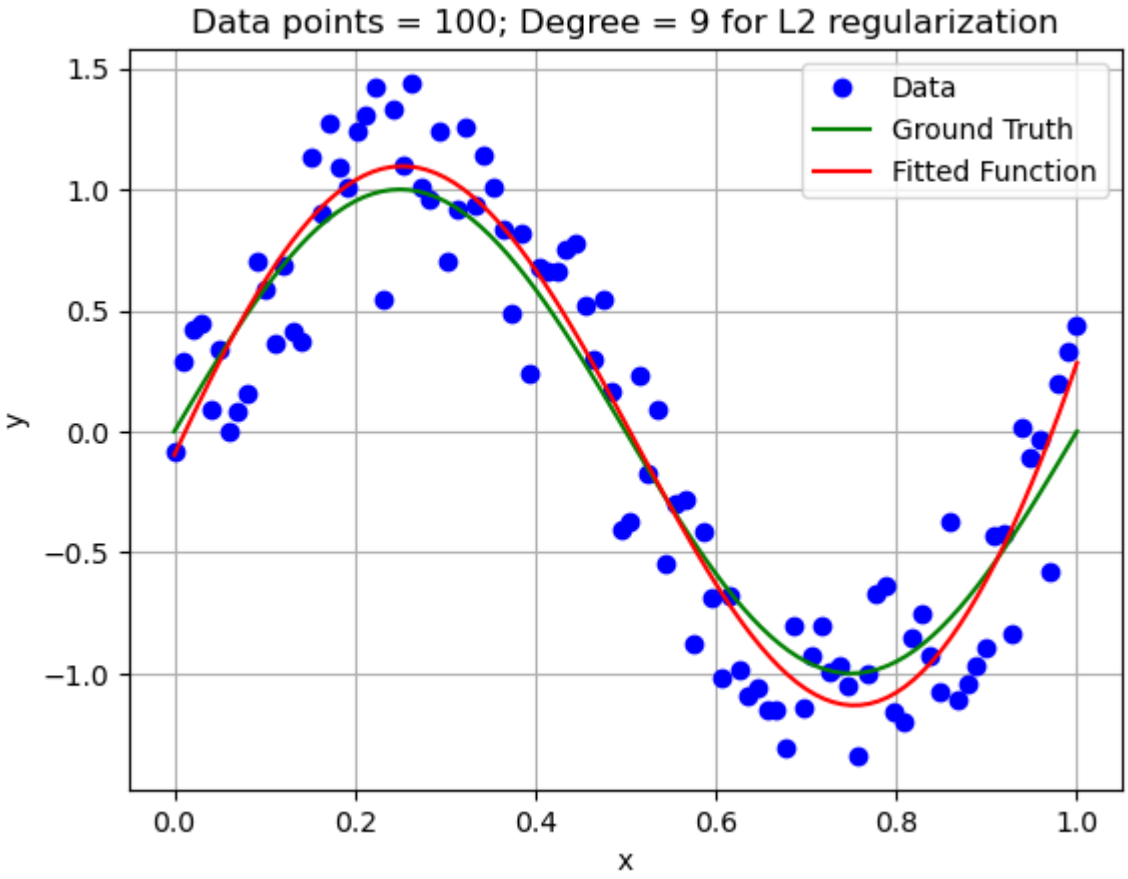
For each model:

- Print the learned model parameters `w`
- Use the model parameters `w` to predict `yr` values over a range of x values given by `xr = np.linspace(0,1,101)`
- Plot the data, ground truth function, and regressed model using `plot_model(x,y,xt,yt,xr,yr,title)` with an appropriate title.

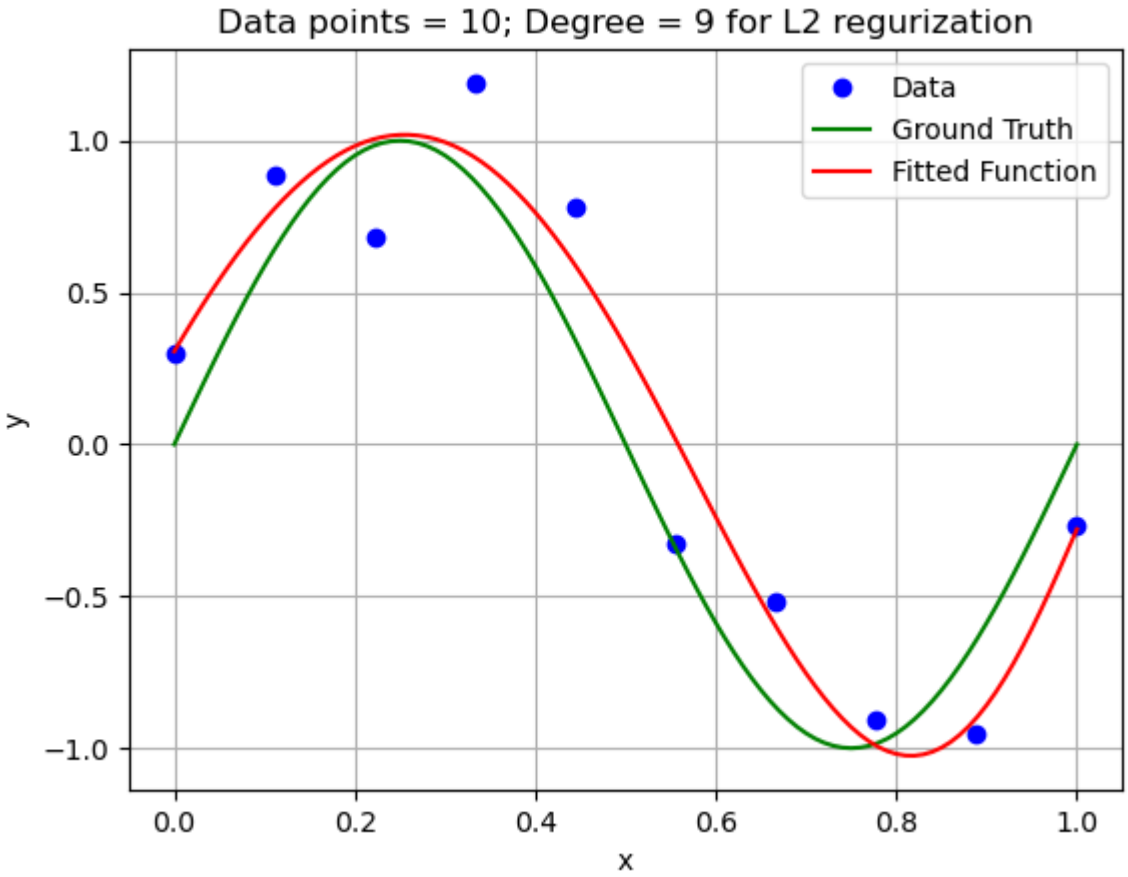
In [81]:
```
# YOUR CODE GOES HERE
def get_w(X,y):
    I_m = np.eye(10)
    I_m[-1,1] = 0
    w = np.linalg.inv(((X.T)@X)+(np.exp(-10)*I_m)) @ (X.T) @ y.reshape(-1,1)
    return w


x100_2 = x100**2
x100_3 = x100**3
x100_4 = x100**4
x100_5 = x100**5
x100_6 = x100**6
x100_7 = x100**7
x100_8 = x100**8
x100_9 = x100**9
X100 = np.array([x100_9, x100_8, x100_7, x100_6, x100_5, x100_4, x100_3, x100_2, x100,

xr100 = np.linspace(0,1,101)
xr100_2 = xr100**2
xr100_3 = xr100**3
xr100_4 = xr100**4
xr100_5 = xr100**5
xr100_6 = xr100**6
xr100_7 = xr100**7
xr100_8 = xr100**8
xr100_9 = xr100**9
xr = np.array([xr100_9, xr100_8, xr100_7, xr100_6, xr100_5, xr100_4, xr100_3, xr100_2,

w100_L2 = get_w(X100,y100)
print("Quadratic Coefficients for degree 9 and 100 data points:\n", w100_L2.flatten(),
yr = xr @ w100_L2
plot_model(x100, y100, xt, yt, xr100, yr, "Data points = 100; Degree = 9 for L2 regula

x10_2 = x10**2
x10_3 = x10**3
x10_4 = x10**4
x10_5 = x10**5
x10_6 = x10**6
x10_7 = x10**7
x10_8 = x10**8
x10_9 = x10**9
X10 = np.array([x10_9, x10_8, x10_7, x10_6, x10_5, x10_4, x10_3, x10_2, x10, np.ones_1
```

```
xr10 = np.linspace(0,1,101)
xr10_2 = xr10**2
xr10_3 = xr10**3
xr10_4 = xr10**4
xr10_5 = xr10**5
xr10_6 = xr10**6
xr10_7 = xr10**7
xr10_8 = xr10**8
xr10_9 = xr10**9
xr = np.array([xr10_9, xr10_8, xr10_7, xr10_6, xr10_5, xr10_4, xr10_3, xr10_2, xr10, r

w10_L2 = get_w(X10,y10)
print("Quadratic Coefficients for degree 9 and 10 data points:\n", w10_L2.flatten(),"\
yr = xr @ w10_L2
plot_model(x10, y10, xt, yt, xr10, yr, "Data points = 10; Degree = 9 for L2 regurizati
```

```
Quadratic Coefficients for degree 9 and 100 data points:
 [ 17.53894662 -22.00648636 -21.06913325   7.03986694  33.82819395
  20.79697103 -38.38292328  -5.4387958    8.07409936  -0.09741016]
```




```
Quadratic Coefficients for degree 9 and 10 data points:
 [-3.53377609 -1.84898314  2.29989838  6.47132847  6.80581128  0.25333561
 -9.30508978 -6.83160954  5.10354376  0.30632365]
```



Discussion:

What differences between the regularized and standard 9th order models fit to `d10` do you notice? How does regularization affect the fitted function?

The L2 regularization gives a better model than the standard 9th order model. The 9th order model leads to overfitting and therefore L2 regularization is better generalizing the data. The L2 regularization leads to a smoother graph with less variance issues. *

LLS with L_2 regularization and gradient descent

For complex models, the size of $X'X$ can be large, making matrix inversion computationally demanding. Instead, one can use gradient descent to compute w. In our notes, we derived the gradient descent approach both for unregularized as well as L_2 regularized linear regression. The formula for the gradient descent approach with L_2 regularization is:

$ \frac{\partial obj}{\partial w} = X'Xw - X'y + \lambda \mathbb{I}_m w$

$ w^{new} \leftarrow w^{cur} - \alpha \frac{\partial obj}{\partial w}$

In this problem, could gradient descent get stuck in a local minimum? Explain why / why not?

Yes it can get stuck in local minimum. Because in L2 regularization the global minima is only one i.e., the local minimum which is also the reason the gradient descent can get stuck in local minima. Also in this case if we take 100 data points then the learning rate becomes too large which can lead to the graph to osscilate thus the gradient descent gets stuck in the local minimum.

You will consider just a single case in the following question:

1. Data: data10.txt, Model: 9th order polynomial.

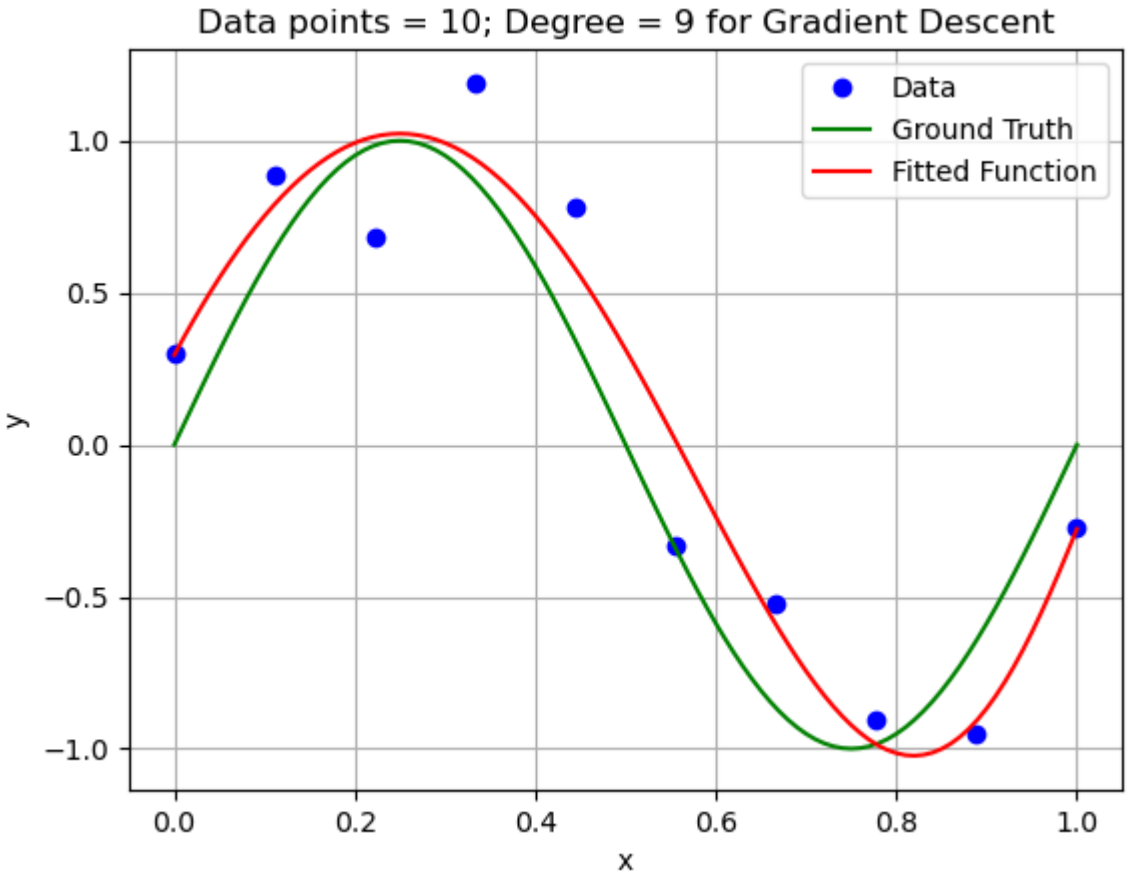
Starting with a weight vector of zeros as the initial guess, and $\lambda = e^{-10}$, $\alpha = 0.075$, apply 50000 iterations of gradient descent to find the optimal model parameters. In practice, when you train your own models you will have to determine these parameters yourself!

For the trained model:

- Print the learned model parameters `w`
- Use the model parameters `w` to predict `yr` values over a range of x values given by `xr = np.linspace(0,1,101)`
- Plot the data, ground truth function, and regressed model using `plot_model(x,y,xt,yt,xr,yr,title)` with an appropriate title.

In [86]:
```
# YOUR CODE GOES HERE
num_iterations = 50000
alpha = 0.075
p_10_gd = PolynomialFeatures(degree = 9)
X_10_gd = p_10_gd.fit_transform(x10.reshape(-1,1))
xr_10_gd = np.linspace(0,1,101)
xr = p_10_gd.fit_transform(xr_10_gd.reshape(-1,1))
w10 = np.zeros(10)

def grad_w(X,y,w):
    I_m = np.eye(10)
    I_m[-1,-1] = 0
    L = np.exp(-10)
    w = X.T@X@w - X.T @y + (L*I_m)@ w
    return w

for i in range(num_iterations):
    w10 = w10 - (alpha*grad_w(X_10_gd,y10,w10))
print("Quadratic coefficients are:", w10)
yr = xr @ w10
plot_model(x10, y10, xt, yt, xr_10_gd, yr, "Data points = 10; Degree = 9 for Gradient
```

```
Quadratic coefficients are: [ 0.29544563  5.56353544 -9.30717072 -5.78373409  0.80366
419  4.60593122
  4.93183109  2.90443341 -0.22697467 -4.06428386]
```



Discussion:

Visually compare the result you just obtained to the same 9th order polynomial model with L_2 regularization where you solved for `w` directly in the previous section. They should be very similar. Comment on whether gradient descent has converged.

The graphs for both of them are very similar but when we talk about L2 regularization it helps in making the graph smoother which in turn can help in reducing teh convergence as the variance also reduces. But when we talk about gradient descent and using that method to train the model. Because the learning rate i.e., number of iterations are too large the graph is converging. Yes the graph converged

In [ ]: